

2025 Activity Report

RESEARCH CENTRE: Inria Paris Centre

IN PARTNERSHIP WITH: CNRS, Ecole normale supérieure de Paris


Project-Team

PARKAS

Synchronous Kahn parallelism



In collaboration with Département d'Informatique de l'Ecole Normale Supérieure



Project-Team PARKAS

Creation of the Project-Team: 2025 June 01

Each year, Inria research teams publish an Activity Report presenting their work and results over the reporting period. These reports follow a common structure, with some optional sections depending on the specific team. They typically begin by outlining the overall objectives and research programme, including the main research themes, goals, and methodological approaches. They also describe the application domains targeted by the team, highlighting the scientific or societal contexts in which their work is situated. The reports then present the highlights of the year, covering major scientific achievements, software developments, or teaching contributions. When relevant, they include sections on software, platforms, and open data, detailing the tools developed and how they are shared. A substantial part is dedicated to new results, where scientific contributions are described in detail, often with subsections specifying participants and associated keywords. Finally, the Activity Report addresses funding, contracts, partnerships, and collaborations at various levels, from industrial agreements to international cooperations. It also covers dissemination and teaching activities, such as participation in scientific events, outreach, and supervision. The document concludes with a presentation of scientific production, including major publications and those produced during the year.

Keywords

Computer sciences and digital sciences

- A1.1.1. – Multicore, Manycore
- A1.1.2. – Hardware accelerators (GPGPU, FPGA, etc.)
- A2.1.1. – Semantics of programming languages
- A2.1.4. – Functional programming
- A2.1.6. – Concurrent programming
- A2.1.9. – Synchronous languages
- A2.1.10. – Domain-specific languages
- A2.2.4. – Parallel architectures
- A2.2.8. – Code generation
- A2.3. – Embedded and cyber-physical systems
 - A2.3.1. – Embedded systems
 - A2.3.2. – Cyber-physical systems
 - A2.3.3. – Real-time systems
 - A2.3.5. – Cyber-physical systems
- A4.5.3. – Program proof
- A6.2.1. – Numerical analysis of PDE and ODE
- A6.4.1. – Deterministic control
- A6.4.2. – Stochastic control

Other research topics and application domains

- B5.2.1. – Road vehicles
- B5.2.2. – Railway
- B5.2.3. – Aviation
- B6.4. – Internet of things
- B6.6. – Embedded systems
- B7.2.1. – Smart vehicles
- B9.5.1. – Computer science
- B9.5.2. – Mathematics

Contents

Project-Team PARKAS	1
1 Team members, visitors, external collaborators	5
2 Overall objectives	5
3 Research program	6
4 Application domains	7
4.1 Embedded Control Software	7
4.2 Hybrid Systems Design and Simulation	7
5 Highlights of the year	7
5.1 Awards	7
6 Latest software developments, platforms, open data	7
6.1 Latest software developments	7
6.1.1 Zelus	7
6.1.2 Vélus	8
6.1.3 presseail	8
6.1.4 SundialsML	9
6.1.5 Heptagon	9
6.1.6 ZRun	10
7 New results	10
7.1 Verified compilation of Lustre	10
7.2 Latency-based scheduling of synchronous programs	11
7.3 Sundials/ML interface for Zelus Runtime	13
7.4 Semantics and Compilation of Arrays in Dataflow Synchronous Languages	13
7.5 The Zelus Language	14
7.6 A Constructive Synchronous Semantics	14
7.7 Translation Validation Techniques for a Synchronous Language Compilers	15
7.8 Verification by Abstract Interpretation of Synchronous Programs	15
7.9 A Functional Semantics for Hybrid System Simulation	15
7.10 What is a zero-crossing?	16
7.11 Zélus with refinement types	16
8 Bilateral contracts and grants with industry	16
8.1 Bilateral contracts with industry	16
9 Partnerships and cooperations	17
9.1 International research visitors	17
9.1.1 Visits of international scientists	17
10 Dissemination	18
10.1 Promoting scientific activities	18
10.1.1 Scientific events: organisation	18
10.1.2 Journal	18
10.1.3 Scientific expertise	18
10.1.4 Research administration	18
10.2 Teaching - Supervision - Juries - Educational and pedagogical outreach	18
10.2.1 Supervision	19
10.2.2 Juries	19
10.3 Popularization	19

10.3.1 Productions (articles, videos, podcasts, serious games, ...)	19
10.3.2 Participation in Live events	19
10.3.3 Others science outreach relevant activities	19
11 Scientific production	20
11.1 Major publications	20
11.2 Publications of the year	21
11.3 Cited publications	21

1 Team members, visitors, external collaborators

Research Scientist

- Timothy Bourke [INRIA, Researcher]

Faculty Members

- Marc Pouzet [Team leader, DI-ENS, Professor, HDR]
- Paul Feautrier [DI-ENS, Emeritus]

PhD Students

- Gregoire Bussone [ENS PARIS]
- Balthazar Patiachvili [ENS PARIS, from Sep 2025]

Technical Staff

- Paul Jeanmaire [INRIA, Engineer, until Aug 2025]
- Loic Sylvestre [INRIA, Engineer, until Aug 2025]

Interns and Apprentices

- Alexandre Douard [INRIA, Intern, from Jun 2025 until Jul 2025]
- Balthazar Patiachvili [ENS PARIS, Intern, from Mar 2025 until Aug 2025]
- Henri Saudubray [INRIA, Intern, from Mar 2025 until Aug 2025]

Administrative Assistants

- Laurence Bourcier [INRIA]
- Eugenie-Marie Montagne [INRIA]

2 Overall objectives

Research in PARKAS focuses on the design, semantics, and compilation of programming languages which allow going from parallel deterministic specifications to target embedded code executing on sequential or multi-core architectures. We are driven by the ideal of a mathematical and executable language used both to program and simulate a wide variety of systems, including real-time embedded controllers in interaction with a physical environment (e.g., fly-by-wire, engine control), computationally intensive applications (e.g., video), and compilers that produce provably correct and efficient code.

The team bases its research on the foundational work of Gilles Kahn on the semantics of deterministic parallelism, the theory and practice of synchronous languages and typed functional languages, synchronous circuits, and formal models to prove the correctness of low-level code.

To realize our research program, we develop languages (LUCID SYNCHRONE, REACTIVEML, LUCY-N, ZELUS), compilers (PRESSEAIL), contributions to open-source projects (Sundials/ML), and formalizations in Interactive Theorem Provers of language semantics and compilers (Vélus). These software projects constitute essential “laboratories”: they ground our scientific contributions, guide and validate our research through experimentation, and are an important vehicle for long-standing collaborations with industry.

3 Research program

Embedded control software is at the heart of many critical applications (medical devices, cars, planes, satellites, trains, energy factories). This complex software interacts in real time with a partially known physical environment and it must ensure statically strong safety constraints (e.g., determinacy, deadlock freedom, execution in bounded time and memory). Model-based design, in its most formal interpretation, is a way to increase confidence that the system and its software behave as expected. It is based on dedicated languages to write executable mathematical specifications (from control theory, physics) which are simulated, tested, formally verified and compiled to executable code. This approach is recognized today by certification authorities and supported by industrial tools and languages like SCADE and SIMULINK.

The compiler plays a central role in these languages: it performs static verification checks on models, source-to-source transformations, optimizations; it generates intermediate representations for formal verification, automatic testing; and finally, executable code for fast simulation and the implementation on an embedded target. Moreover, for hybrid systems modeling languages (e.g., Simulink, Modelica), the compiler does complex static analyses and transformations (e.g., index reduction, differentiation, computation of the Jacobian) in order to produce simulation code. The run-time system — the so-called “simulation engine” — which involves numeric solvers for differential equations and zero-crossing detection is itself a complex software whose correctness is critical to get confidence in simulation results. How to guarantee that all these compilation steps and the run-time system are correct? That what is simulated, tested and verified corresponds to what is written in the source program and remains true of the generated code? In other words, that we have a *high fidelity chain going from a mathematically precise specification to its faithful reproduction on a target architecture* or, to paraphrase Gérard Berry, that “what you simulate/prove on the model is what you execute” [19].

The scientific objective of PARKAS is to tackle this fundamental question, making concrete proposals in the form of language and compiler prototypes, to improve the best industrial practices for certified applications where SCADE is the state-of-the-art. We aim at extending the theory and practice on the design, semantics and implementation of *domain-specific languages for reactive systems*, allowing to go from a *precise specification down to trustable code*, with a *precise and traceable compilation chain* that gives strong and explainable evidence of correctness.

Our research draws its inspiration and focus from the simplicity and complementarity of the data-flow and deterministic model of parallelism introduced by Kahn and Mac Queen [33, 34], the theory and practice of synchronous programming,[30, 18, 27] typed functional programming [40] and the fruitful relationships between the two.[41, 42, 24, 36, 17] To reach our goal, we leverage a large body of formal principles: language and compiler design, semantics, dedicated type systems, proof engineering for compilers, synchronous circuits, and compilation algorithms.

Our research program has recently been organized along two axes.

1. The definition of languages for cyber-physical system allowing to write hybrid (discrete/continuous) models (e.g., the interaction between software and a physical environment). We have investigated how to express and exploit relaxed model of synchrony such as “communication by sampling” through bounded buffers; models that mix reactive control and array-based computer intensive applications; probabilistic programming constructs to model uncertainty; and random testing techniques for finding bugs.
2. An important and closely interconnected challenge is the ability to preserve the trustworthiness of a compiler chain, giving the greatest confidence in its correctness, from the model to the code. We develop precise specifications (on paper) for all the language extensions we propose. When the solution is mature enough, we develop computer-aided and machine-checked formal semantics and compilation techniques. We have treated important language extensions (e.g., the mix of data-flow and hierarchical automata). We have also explored two executable denotational semantics. One is stream-based (or “Kahnian”); the other is state-based and lead to an effective interpreter.

In our research, we adopt and follow a *language-centric approach*, focusing our efforts on the development of concrete languages proposals, dedicated type systems, compile-time static analyses, compilation algorithms, etc. These softwares constitute essential “laboratories”: they ground our scientific contributions, guide

and validate our research through experimentation, and they are an important vehicle for teaching and long-standing collaborations with industry. ZELUS, VÉLUS, ZRUN, PRESSEAIL are visible results of our approach.

4 Application domains

4.1 Embedded Control Software

Embedded control software defines the interactions of specialized hardware with the physical world. It normally ticks away unnoticed inside systems like medical devices, trains, aircraft, satellites, and factories. This software is complex and great effort is required to avoid potentially serious errors, especially over many years of maintenance and reuse.

Engineers have long designed such systems using block diagrams and state machines to represent the underlying mathematical models. One of the key insights behind synchronous programming languages is that these models can be executable and serve as the base for simulation, validation, and automatic code generation. This approach is sometimes termed Model-Based Development (MBD). The SCADE language and associated code generator allow the application of MBD in safety-critical applications. They incorporate ideas from LUSTRE, LUCID SYNCHRONE, and other programming languages.

4.2 Hybrid Systems Design and Simulation

Modern embedded systems are increasingly conceived as rich amalgams of software, hardware, networking, and physical processes. The terms Cyberphysical System (CPS) or Internet-of-Things (IoT) are sometimes used as labels for this point of view.

In terms of modeling languages, the main challenges are to specify both discrete and continuous processes in a single *hybrid* language, give meaning to their compositions, simulate their interactions, analyze the behavior of the overall system, and extract code either for target control software or more efficient, possibly online, simulation. Languages like Simulink and Modelica are already used in the design and analysis of embedded systems; it is more important than ever to understand their underlying principles and to propose new constructs and analyses.

5 Highlights of the year

5.1 Awards

- Timothy Bourke received an *Outstanding Reviewer Award* at the 21st ACM/IEEE Embedded Systems Week in Taipei, Taiwan for the EMSOFT conference.
- Timothy Bourke received a *Distinguished Reviewer Award* at the 20th International Conference on Integrated Formal Methods, iFM 2025, in Paris, France.

6 Latest software developments, platforms, open data

6.1 Latest software developments

6.1.1 Zelus

Keywords: Numerical simulations, Compilers, Embedded systems, Hybrid systems

Scientific Description: The Zélus implementation has two main parts: a compiler that transforms Zélus programs into OCaml programs and a runtime library that orchestrates compiled programs and numeric solvers. The runtime can use the Sundials numeric solver, or custom implementations of well-known algorithms for numerically approximating continuous dynamics.

Functional Description: Zélus is a new programming language for hybrid system modeling. It is based on a synchronous language but extends it with Ordinary Differential Equations (ODEs) to model continuous-time behaviors. It allows for combining arbitrarily data-flow equations, hierarchical automata and ODEs. The language keeps all the fundamental features of synchronous languages: the compiler statically ensure the absence of deadlocks and critical races, it is able to generate statically scheduled code running in bounded time and space and a type-system is used to distinguish discrete and logical-time signals from continuous-time ones. The ability to combines those features with ODEs made the language usable both for programming discrete controllers and their physical environment.

URL: <https://zelus.di.ens.fr>

Publications: [hal-03051954v1](#), [hal-02333603v1](#), [hal-02426533v1](#), [inria-00554271v1](#), [hal-01242732v1](#), [hal-00654113v1](#), [hal-00909029v1](#), [hal-01575621v4](#), [hal-01575631v1](#), [hal-00766726v1](#), [hal-00938891v1](#), [hal-00654112v1](#), [hal-01879026v1](#), [hal-01549183v2](#), [hal-00938866v1](#)

Contact: Marc Pouzet

Participants: Marc Pouzet, Timothy Bourke

Partner: ENS Paris

6.1.2 Vélus

Name: Verified Lustre Compiler

Keywords: Synchronous Language, Compilation, Software Verification, Coq, Ocaml

Functional Description: Vélus is a prototype compiler from a subset of Lustre to assembly code. It is written in a mix of Coq and OCaml and incorporates the CompCert verified C compiler. The compiler includes formal specifications of the semantics and type systems of Lustre, as well as the semantics of intermediate languages, and a proof of correctness that relates the high-level dataflow model to the values produced by iterating the generated assembly code.

Release Contributions: Vélus 3.0 introduces syntax and semantics for Lustre (previous versions only treated the normalized form of Lustre). It includes a verified normalization pass that transforms Lustre programs into NLustre programs.

URL: <https://velus.inria.fr>

Publications: [hal-01817949](#), [hal-03287572](#), [hal-01512286](#), [hal-01403830](#), [tel-03068862](#), [hal-02005639](#), [hal-02426573](#), [hal-03370264](#)

Contact: Timothy Bourke

Participants: Timothy Bourke, Basile Pesin, Paul Jeanmaire, Marc Pouzet

6.1.3 presseail

Name: All-in-Lustre Compiler

Keywords: Embedded systems, Compilers, Synchronous Language, Real-time application

Functional Description: The input to the compiler is the rate-synchronous language described in our ECRTS 2023 article. The compiler generates and Integer Linear Programming (ILP) problem that includes data dependency and resource constraints. The problem is solved using an external solver and the resulting schedule is used by the compiler to generate sequential code using a generalization of the modular clock-driven compilation scheme used in modern Lustre/Scade compilers. The compiler implements special features for analyzing and eliminating cyclic data dependencies.

Release Contributions: First version described in the ECRTS 2023 publication.

Contact: Timothy Bourke

6.1.4 SundialsML

Name: Sundials/ML

Keywords: Simulation, Mathematics, Numerical simulations

Scientific Description: Sundials/ML is a comprehensive OCaml interface to the Sundials suite of numerical solvers (CVODE, CVODES, IDA, IDAS, KINSOL). Its structure mostly follows that of the Sundials library, both for ease of reading the existing documentation and for adapting existing source code, but several changes have been made for programming convenience and to increase safety, namely: solver sessions are mostly configured via algebraic data types rather than multiple function calls, errors are signalled by exceptions not return codes (also from user-supplied callback routines), user data is shared between callback routines via closures (partial applications of functions), vectors are checked for compatibility (using a combination of static and dynamic checks), and explicit free commands are not necessary since OCaml is a garbage-collected language.

Functional Description: Sundials/ML is an OCaml interface to the Sundials suite of numerical solvers (CVODE, CVODES, IDA, IDAS, KINSOL, ARKODE).

Release Contributions: Sundials/ML v6.0.0p0 adds support for v5.x and v6.x of the Sundials Suite of numerical solvers. This includes the latest Arkode features, many vectors, and nonlinear solvers.

URL: <http://inria-parkas.github.io/sundialsml/>

Publications: [hal-01408230v1](#), [hal-01967659v1](#)

Contact: Timothy Bourke

Participants: Jun Inoue, Marc Pouzet, Timothy Bourke

6.1.5 Heptagon

Keywords: Compilers, Synchronous Language, Controller synthesis

Functional Description: Heptagon is an experimental language for the implementation of embedded real-time reactive systems. It is developed inside the Synchronics large-scale initiative, in collaboration with Inria Rhones-Alpes. It is essentially a subset of Lucid Synchronic, without type inference, type polymorphism and higher-order. It is thus a Lustre-like language extended with hierarchical automata in a form very close to SCADE 6. The intention for making this new language and compiler is to develop new aggressive optimization techniques for sequential C code and compilation methods for generating parallel code for different platforms. This explains much of the simplifications we have made in order to ease the development of compilation techniques.

The current version of the compiler includes the following features: - Inclusion of discrete controller synthesis within the compilation: the language is equipped with a behavioral contract mechanisms, where assumptions can be described, as well as an "enforce" property part. The semantics of this latter is that the property should be enforced by controlling the behaviour of the node equipped with the contract. This property will be enforced by an automatically built controller, which will act on free controllable variables given by the programmer. This extension has been named BZR in previous works. - Expression and compilation of array values with modular memory optimization. The language allows the expression and operations on arrays (access, modification, iterators). With the use of location annotations, the programmer can avoid unnecessary array copies.

URL: <https://gitlab.inria.fr/synchrone/heptagon>

Contact: Gwenaël Delaval

Participants: Gwenaël Delaval, Marc Pouzet, 5 anonymous participants

Partners: UGA, ENS Paris, Inria, LIG

6.1.6 ZRun

Name: The ZRun Synchronous Language Interpreter

Functional Description: ZRun is an executable semantics of a synchronous data-flow language. It takes the form of a purely functional interpreter and is implemented in OCaml. The input of Zrun is a large subset of the language Zélus, but only its discrete-time (synchronous) subset. The basic primitives are those of Lustre: a unit non-initialized delay (pre), the initialization operator (->), the initialized delay (fby), and streams can be defined by mutually recursive definitions. It also provides richer programming constructs that were introduced in Lucid Synchrone and Scade 6, but are not in Lustre: the by-case definition of streams, the last computed value of a signal, hierarchical automata with parameters, stream functions with static parameters that are either known at compile time or at instantiation time, and two forms of iterators on arrays: the "forward" to perform an iteration in time, the "foreach" to perform an iteration on space.

The objective of this prototype is to give a reference executable semantics that is independent of a compiler. It can be used, e.g., as an oracle for compiler testing, to execute unfinished programs or programs that are semantically correct but are statically rejected by the compiler.

Release Contributions: Branch Master (2000) - v1.x. - first-order language, streams, hierarchical automata, by-case definition of streams, operator last.

Branch Works (2023): - v2.x - static higher-order, hierarchical automata with parameters, valued signals. - arrays, - "forward" and "foreach" iterations.

URL: <https://github.com/marcpouzet/zrun>

Contact: Marc Pouzet

7 New results

7.1 Verified compilation of Lustre

Participants: Timothy Bourke, Paul Jeanmaire, Balthazar Patiachvili, Marc Pouzet.

Vélus is a compiler for a subset of LUSTRE and SCADE that is specified in the Coq [28] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [35, 20] to define the semantics of machine operations (integer addition, floating-point multiplication, etcetera) and to generate assembly code for different architectures. The research challenges are to

- to mechanize, i.e., put into Coq, the semantics of the programming constructs used in modern languages for Model-Based Development;
- to implement compilation passes and prove them correct;
- to interactively verify source programs and guarantee that the obtained invariants also hold of the generated code.

Work continued this year on this long-running project by continuing to develop a functional model for interactive verification, and by experimenting with verified node inlining. The details are described below. After his M2 internship, Balthazar Patiachvili joined the team as a PhD student working on arrays in verified compilers for dataflow synchronous languages.

Functional semantics for program verification: To date we have focused on proving the correctness of compilation passes. This involves specifying semantic models to define the input/output relation associated with a program, implementing compilation functions to transform the syntax of a program, and proving that the relation is unchanged by the functions. In addition to specifying compiler correctness, semantic models can also serve as a base for verifying individual programs. The challenge is to present and manipulate such detailed specifications in interactive proofs. The potential advantage is to be able to reason on abstract models and to obtain, via the compiler correctness theorem, proofs that apply to generated code. Making this idea work requires solving several scientific and technical challenges. It was the subject of Paul Jeanmaire’s thesis, defended in late 2024.

An article based on the material of Paul Jeanmaire PhD [32] was accepted and presented at the LICS conference [14]. This work describes a constructive denotational model for the dataflow subset of Lustre/Scade. It models errors explicitly and includes a proof in Rocq that the functional model satisfies the relational predicates used in the compiler verification. This gives both a formal link to the generated code and confidence in the correctness of the relational predicates. We continued working to better understand and document the underlying library on Complete Partial Orders (CPOs) [38].

The Scade 6 language also includes control structures, and notably hierarchical state machines [26, 25] that were previously formalized in the relational model [39, 23] but not treated in Paul Jeanmaire’s PhD. We made a first step toward adding this feature to Vélus by extending an existing Kahn semantics using lazy streams in OCaml and then adapting it to the CPO-based model in Coq. This works quite well, but becomes complicated for state machines with *entry-by-history*. The results will be presented at the JFLA in early 2026.

Verified Node Inlining: Inlining refers to the replacement of a function call by the contents of the function. It is a typical compiler optimization that is, for instance, implemented and verified in the CompCert compiler [35]. For a dataflow synchronous language, the term “node instance” is used to emphasize that the problem involves resettable stream functions which are ultimately compiled to step functions with explicit state. There are two reasons for adding a node-inlining pass to a compiler like Vélus. First, it provides one way of compiling node instances involved in feedback loops. Second, inlining early may give more optimization opportunities to downstream compiler passes.

In Vélus, inlining could be implemented in the source *Lustre* language or the normalized *NLustre* language. The advantage of inlining in Lustre is that it provides block-based constructions for modular resets and local variables, and the related compilation passes are already implemented and verified [39]. The advantage of inlining in NLustre is that its syntax and semantics is more structured and thus simpler to work with in the proof assistant; notably, the induction schemes are less sophisticated. In the M2 internship of Balthazar Patiachvili, we decided to concentrate on node inlining in NLustre.

The inlining pass was not difficult to implement. Verifying it involves showing that a number of syntactic properties, e.g., typing and clock typing, are preserved. This is tedious and time consuming but not especially difficult. Verifying semantic preservation is, however, more difficult. We reduced the problem to one of inversion in the presence of reset and managed to prove that it holds. We did not have time, however, to reestablish the end-to-end proof. This experience suggests that it may be better to implement inlining in the Lustre language. The increased difficulty of induction over the block-based constructs may be outweighed by easier semantic preservation proofs.

Glossary

Interactive Theorem Prover (ITP, also known as a *proof assistant*): Software for formal specification and proof, with features for generating and checking proofs, and extracting programs for later compilation

Model-Based Development (MBD): The specification of control software using block-diagrams, state machines, and other high-level constructions allowing programmers to focus on describing desired behaviour and to rely on automatic code generation to produce low-level executables.

7.2 Latency-based scheduling of synchronous programs

Participants: Timothy Bourke, Marc Pouzet, Loïc Sylvestre.

External collaborators: Dumitru Potop Butucaru (Inria) and; Matthieu Boitrel, Marc Brundler, Matthieu David, Victor Jegu, Sylvain Sauvart, and Jean Souyris (Airbus).

Teams at Airbus have begun planning for the next generation of flight control software. One of their main aims is to enhance the existing development process in two related ways: (i) by providing new specification mechanisms that permit control engineers to express application requirements more directly and with more flexibility; (ii) by exploiting these mechanisms during code generation to increase automation and better exploit existing single-core and future multi-core computing units. We have been working with them on language design and compilation techniques for large synchronous specifications.

In an approach termed “All-in-Lustre”, the top-level node of a Lustre program is distinguished from inner nodes. It contains special annotations to specify the triggering and other details of node instances from which separate “tasks” are to be generated. Special operators are introduced to describe the buffering between top-level instances. Notably, different forms of the `when` and `current` operators are provided. Some of the operators are under-specified and a constraint solver is used to determine their exact meaning, that is, whether the signal is delayed by zero, one, or more cycles of the receiving clock, which depends on the scheduling of the source and destination nodes. Scheduling is formalized as a constraint solving problem based on latency constraints between some pairs of input/outputs that are specified by the designer.

This year, we continued experimenting with including the allocation of tasks to cores within generated ILP constraints. In our approach, inter-core communications within a single cycle are forbidden, and additional synchronization instructions are thus not needed. By improving the *same thread or different phase* encoding and making better use of ILP-solver features, we managed to schedule the largest case study in a more reasonable amount of time.

With Loïc Sylvestre, we started experimenting with the CP-SAT solver included with Google’s OR Tools. Since our problem is increasingly more combinatorial than linear, it is natural to look at SAT-based constraint solvers. We experimented with a number of prototypes: converting the LP format directly into a Python script, exploiting specific constraints for some forms, and eliminating integer variables. Converting the LP format directly with some specific constraints gives quite promising results. Despite time invested into eliminating integer variables, studying pseudo-boolean constraint encodings [29], and experimenting with MiniSat+, it is difficult to beat the lazy constraint generation of CP-SAT. We added more schedule validation within the compiler, notably checking that dependency constraints are respected by external scheduling. The CP-SAT solver can be tricky to compile and install, so we started developing an opam package to make it a more reliable dependency for projects in OCaml.

For programs with many tasks running at the fastest rate, the *same thread or different phase* approach is insufficient to obtain good load balancing, since it prevents separating some producer-consumer pairs across different cores. We implemented a *-base-barrier* option that inserts an additional synchronization barrier and allows the ILP solver to place fast tasks before or after the barrier. This increases the combinatorial complexity of the constraint problem, but gives better load balancing on the case-studies that we examined.

The constraint generation module in the `presseail` compiler had become too complicated, so we restructured it to remove clustering (which ultimately proved ineffective) and simplify the treatment of variables. We also included sample choices in the internal flowgraph representation to facilitate the removal of redundant constraints, better validate schedules, and potentially allow for a finer-grained causality analysis. We also introduced an explicit model for “unconstrained concomitance” and reworked options for eliminating cycles.

We presented our work in a special-session on end-to-end latency in Cyber Physical systems at the EMSOFT conference [16] and started work on a joint proposal with collaborators in Germany.

Together with our collaborators at Airbus and support from the transfer (Estelle Gaspard), legal (Manon Coger), and financial (Odette Dabire) services of Inria Paris, we contributed to a project proposal on mixing control algorithms and matrix manipulations within a dataflow synchronous language.

This work is funded by direct industrial contracts with Airbus.

Glossary

Integer Linear Programming (ILP): A problem is encoded as a list of linear constraints on real and integer variables, together with a linear goal expression. Solver software attempts to find values for the variables such that the constraints are satisfied while minimizing or maximizing the value of the goal expression.

7.3 Sundials/ML interface for Zelus Runtime

Participants: Timothy Bourke, Alexandre Douard, Marc Pouzet.

Sundials [31] is a collection of six numeric solvers. We developed an OCaml interface to this library [22, 21] to better understand it and to integrate it into the runtime of the Zelus programming language. This interface has been more difficult to maintain than expected due to changes in OCaml 5.x, to introduce the multi-threaded runtime, and to the Sundials library itself, especially around the ARKode solver.

Previously, we reported a behavior change in the OCaml 5.x runtime which caused deadlocks in some of the Sundials/ML examples. The suggested workaround was not completely satisfying, but this year the OCaml runtime itself was changed to avoid the problem we identified. This allowed us to continue our work on updating the interface. We first treated Sundials versions 6.6.0 and 6.7.0. Then, with Alexandre Douard, during his summer L3 internship, we treated version 7.0.0 and almost completed 7.1.0. Alexandre Douard added a test harness and worked especially on the ARKode interface.

7.4 Semantics and Compilation of Arrays in Dataflow Synchronous Languages

Participants: Timothy Bourke, Grégoire Bussone, Marc Pouzet.

The thesis project of Gregoire Bussone studies the treatment of arrays and state machines in synchronous languages, with an emphasis on the generation of fast and memory-efficient code, and eventual verification in a proof assistant. The problem is essentially to transform a functional input language where modifying an array element creates a new array into sequential code with in-place updates wherever possible. State machines are interesting in this respect, due to the exclusive-or nature of states: when states are always reset on entry, they can share the same underlying memory. Similarly, if a loop over an array is reset at each instant, intermediate arrays need not be stored as state variables.

This year we studied the link between memory allocation and scheduling. There are trade-offs between scheduling flexibility, copy minimization, and compiler complexity. Certain operations, like array concatenation and slices require special treatment. We proposed an intermediate language that takes this into account and defined its type systems and semantic models. The MADL language, developed during Baptiste Pauget's PhD [37], addresses some of these aspects but not node instances or conditional activation (clocks).

We also worked on *views*, which allow some array operations to be implemented as calculations on indexes rather than memory modifications. For example, the `reverse` function can be implemented by transforming an index i into $size - i$. This optimization is difficult to apply in a modular manner. We experimented with a compilation scheme that exploits the *traits* of Rust to encode the index transformations. The intermediate code requires a certain amount of technical annotations, but the resulting assembly code is compact and fast.

We worked on the *forward* and *foreach* loops that allow mixing dataflow and array operators by alternating between streams of arrays and arrays of streams. We also worked on a new intermediate form for state machines that generalizes the treatment of strong and weak transitions, and incorporates a static analysis based on when states are reset to identify opportunities for sharing storage.

7.5 The Zelus Language

Participants: Timothy Bourke, Marc Pouzet, Gregoire Bussone.

Zelus is our laboratory to experiment our research on programming languages for hybrid systems. It is devoted to the design and implementation of systems that may mix discrete-time/continuous-time signals and systems between those signals. It is first a synchronous language reminiscent of Lustre and Lucid Synchrone with the ability to define functions that manipulate continuous-time signals defined by Ordinary Differential Equations (ODEs) and zero-crossing events. The language is functional in the sense that a system is a function from signals to signals (not a relation). It provides some features from ML languages like higher-order and parametric polymorphism as well as dedicated static analyses.

Distribution of the language The language, its compiler and examples (release 2.1) are on [GitHub](#). It is also available as an OPAM package. All the installation machinery has been greatly simplified.

The implementation of Zelus is now relatively mature. The language has been used in a collection of advances projects; the most important of the recent years being the design and implementation of ProbZelus on top of Zelus. This experiment called for several internal deep changes in the Zelus language.

One of the biggest troubles we faced when implementing Zélus was the lack of a tool to automatically test the compiler and to prototype language extensions before finding how to incorporate in the language and how to compile them. This is what motivated first our work on an *executable* semantics. The tool *Zrun* works well now. It is detailed in the Section below. Based on it, we have started a new implementation of Zélus with the objective that every pass of the compiler can be tested, using *Zrun* as an oracle.

In 2024, we have started a new implementation of Zélus and its compiler (see the current development at [the Zélus repository](#)). The main new features are the following:

- The input language now include new programming constructs: functions parameterized by (integer) sizes that are ultimately known at compile-time. Size expressions can appear in the type of arrays (e.g., iterators) and to define recursive functions on sizes (e.g., FFT, recursive search by dichotomy). Two important new constructs have been added : the forward loop construct which iterate a stream function on an array (cf. PhD of Baptiste Pauget) and the foreach loop constructs which corresponds to running several stream functions in parallel (e.g., also called "multi-instanciation).
- The type system and compilation has been extended to deal with those new constructs. Through the techniques are classical and well understood (essentially, a simple variant of HM(X)), it have demanded quite an important work.
- The compiler is now paired with ZRun which is used as an oracle for (black-box, random) testing of the compiler.

We simplified and reorganised several parts of the compiler; e.g., a new implementation of the typing phrase, a new way of implementing the sequence of source-to-source transformations for generating sequential code; new static analyses; and the pairing with ZRun for testing the compiler. We also wanted that the compiler organisation to be easier for other to use it for their research and experiment with new ideas.

One promising extension is driven by Prof. Jean-Baptiste Jeannin (Univ. Michigan, Ann Arbor), who is spending a sabbatical year at Parkas (June 2024-June 2025). With his students, he develops MarVeLus, an extension of Zélus with refinement types. We collaborate on the extension of Zélus to integrate a new typing pass for formally verifying safety properties as types and expressed by synchronous observers.

7.6 A Constructive Synchronous Semantics

Participants: Baptiste Pauget, Marc Pouzet.

External collaborators: Jean-Louis Colaco (ANSYS, Toulouse, France); Michael Mendler (Univ. of Bamberg, Germany).

In 2024, we have continued the development of ZRun. We develop it in parallel with Zélus such that every new programming construct (and more generally, any language features) is implemented in both the compiler and the interpreter.

The semantics is implemented as an interpreter in a purely functional style, in OCaml. The source code of this development is available at [the Zrun repository](#).

7.7 Translation Validation Techniques for a Synchronous Language Compilers

Participants: Timoty Bourke, Grégoire Bussone, Marc Pouzet.

Grégoire Bussone stated his PhD. in April 2023. He studies the use of translation validation techniques applied to a realistic synchronous language compiler. The objective is to deal with the compilation of array operations and, more generally, memory location. Arrays are not supported in Vélus for the moment. The problem is difficult and occurs in two situations: avoid copies for functional iterators (e.g., map, fold, transpose, concat, reverse); optimize the representation of the state in the final target code (e.g., C) and avoid useless copies for states whose lifetime never intersect (a classical situation that comes for a Scade-like hierarchical automaton where all states are entered by reset). For this work, we follow a translation validation approach, relying on an untrusted compiler and an independent but trustable validation step. We also target a richer and type-safe language back-end (here Rust) instead of C to transmit some of the invariants from the source. In the longer term, the purpose is to be able to implement and to machine-check the correctness of compilation techniques for a synchronous language with arrays and their efficient compilation.

During year 2023, several compilation steps that are implemented in the Zélus compiler have been implemented as translation validation functions proved correct in Coq, notably the inlining, renaming, scheduling, normalization. Internally, the technique employs the "locally nameless representation" introduced by Chargueraud. The input language is, for the moment, a simple subset of Zélus. The treatment of MADL is under way.

7.8 Verification by Abstract Interpretation of Synchronous Programs

Participants: Marc Pouzet, Charles De Haro, Xavier Rival.

In Sept. 2024, Charles de Haro have started his PhD. thesis (Dir. Xavier Rival, INRIA project team Antique; Marc Pouzet, INRIA project team Parkas) on the formal verification of synchronous programs. The objective is to verify safety properties for programs that mix data-flow equations, hierarchical automata and arrays. Those constructs are present in the industrial language Scade and the languages and compilers developed at Parkas, namely Zélus and the Vélus compiler.

While previous works (in particular, Bertrand Jeannot PhD. thesis, in 2000) have considered a subset of the language Lustre (a purely data-flow subset, without clocks), the objective here is to deal with the mix with control structures, like the by-case definition of streams and hierarchical automata, exploiting structural informations that is present in the source. During first year, Charles have defined a new abstract semantics, reminiscent of the concrete, state-based semantics presented at Emsoft'23 and implemented in the ZRun interpreter.

7.9 A Functional Semantics for Hybrid System Simulation

Participants: Marc Pouzet, Henri Saudubray.

Modeling languages for hybrid systems rely on a specific runtime called a "simulation loop," which uses two auxiliary software components: a solver for the numerical resolution of ordinary differential equations written in the Zélus source code, and a zero-crossing detection algorithm. This loop is challenging to implement because, for the simulation to be as efficient as possible, it is necessary to avoid copying arrays (which contain the velocities and positions defined in the model) and to prevent unnecessary reinitializations of the solver.

With Henri Saudubray (M2 intern at MPRI), we developed a precise formalization of this simulation loop, described as a Mealy machine in a purely functional style (a pair $s:S$ and a transition function of type $f: S \rightarrow A \rightarrow B * S$). This leads to a first, purely functional description, parameterized by the integration function given by the numerical solver and the zero-crossing detection function. The second version is an imperative form deduced from the purely functional one where state modifications are done in place. This leads to an effective implementation to be used, in the longer term, as the back-end for Zélus. Ultimately, we want this simulation loop semantics to be added to the ZRun interpreter in order to have an interpreter that handles the entirety of Zélus, including the hybrid part, and that is independent of the compiler.

7.10 What is a zero-crossing?

Participants: Marc Pouzet, Jean-Baptiste Jeannin, Jiawei Chen, Serra Dane.

Most hybrid systems modeling languages (e.g., Simulink, Modelica), an Zélus is no exception rely on a zero-crossing detection algorithm. But what exactly does it mean for a signal to cross zero, regardless of the algorithm that detects it, which may not be complete (i.e., it may not detect that a signal crossed zero between two time points)? This year, we investigated this question by proposing a mathematical definition independent of the algorithm used (in the case of Zélus, we use a classical algorithm, the Illinois method). This work has led to a publication that will appear at NFM 2026 (NASA Formal Methods Symposium).

7.11 Zélus with refinement types

Participants: Marc Pouzet, Jean-Baptiste Jeannin, Jiawei Chen, Serra Dane.

The field of functional programming languages has developed significant work on integrating dependent types by using a decision algorithm to type programs. This allows properties of the program to be expressed in the types and to ensure that the program respects them modularly. Several functional languages have integrated these "refinement types," notably, Liquid Haskell and F*.

Jean-Baptiste Jeannin and his students aim to develop a version of the Zélus language with "refinement types." We are collaborating on this project. During the year, Jiawei Chen, a doctoral student completing his thesis on the subject, visited the team PARKAS several times. Work is underway to define typing rules for mutually recursive stream definitions and hierarchical automata, based on the co-iterative semantics (EMSOFT'23). Some preliminary works has been done for a simple case of Zélus programs with ordinary differential equations. This part will be presented this year, at NFM'26.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

Collaboration with Airbus

Participants: Timothy Bourke, Marc Pouzet.

Our work on rate-synchronous programs is funded by contracts with Airbus.

9 Partnerships and cooperations

9.1 International research visitors

9.1.1 Visits of international scientists

Other international visits to the team

Participants: Marc Pouzet, Timothy Bourke.

Jean-Baptiste Jeannin

Status Assistant Professor

Institution of origin: University of Michigan — Ann Arbor

Country: USA

Dates: 9/2024–8/2025

Context of the visit: Collaboration on semantics and verification of embedded systems

Mobility program/type of mobility: sabbatical

Oleg Kiselyov

Status Professor

Institution of origin: Tohoku University

Country: Japan

Dates: 09/2025

Context of the visit: Collaboration on meta programming

Mobility program/type of mobility: research stay

Mary Sheeran

Status Professor

Institution of origin: Chalmers University of Technology

Country: Sweden

Dates: 12/2025

Context of the visit: Collaboration on circuit design and verification

Mobility program/type of mobility: research stay

John Hughes

Status Professor / CEO

Institution of origin: Chalmers University of Technology / Quviq AB

Country: Sweden

Dates: 12/2025

Context of the visit: Collaboration on testing techniques

Mobility program/type of mobility: research stay

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the conference program committees

- Timothy Bourke served on the PC of ECRTS 2025, the 37th Euromicro Conference on Real-Time Systems.
- Timothy Bourke served on the PC of EMSOFT 2025, the 25th International Conference on Embedded Software.
- Timothy Bourke served on the PC of DATE 2025 (track E1: Embedded software architecture, compilers and tool chains), Design, Automation and Test in Europe Conference.
- Timothy Bourke served on the PC of iFM 2025, the 20th International Conference on Integrated Formal Methods.
- Timothy Bourke served on the PC of RTSOPS Workshop 2025, the 13th International Real-Time Scheduling Open Problems Seminar.
- Timothy Bourke served on the PC of SETTA 2025, the Symposium on Dependable Software Engineering: Theories, Tools and Applications.

10.1.2 Journal

Member of the editorial boards

- Timothy Bourke is a guest editor for the SETTA 2024 special edition of the Journal of Systems Architecture (JSA).

Reviewer - reviewing activities

- Timothy Bourke reviewed articles for ACM Transactions on Embedded Computing Systems (TECS).

10.1.3 Scientific expertise

- Timothy Bourke reviewed project submissions for the ANR AAPG 2025.
- Timothy Bourke reviewed project submissions for the Conseil Franco-Québécois de Coopération Universitaire (CFQCU) programme Samuel de Champlain 2026-2027.

10.1.4 Research administration

- Timothy Bourke served on the Inria Paris CRCN/ISFP commission.
- Timothy Bourke participated in five Thesis Tracking Committees (CSI).

10.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

- Marc Pouzet is Director of Studies for the CS department, at ENS.
- Marc Pouzet and Timothy Bourke: “Parallélisme synchrone” au Master parisien de recherche en informatique (MPRI; M2)
- Licence : Timothy Bourke: “Operating Systems” (L3), Lectures and TDs, ENS, France.
- Master: Marc Pouzet: “Synchronous Reactive Languages” (M2), Lectures, Master CPS (Cyber-physical Systems, led by Sergio Mover (École Polytechnique).

- Master: Marc Pouzet "The Elements of Computing Systems". Cycle pluridisciplinaire d'études supérieures (CPES), L2.
- Master: Timothy Bourke: "A Programmer's introduction to Computer Architectures and Operating Systems" (M1), École Polytechnique, France
- Master: Timothy Bourke presented lectures and TPs on Synchronous Languages in Carlos Agon's course on concurrent models at Sorbonne Université.
- Bachelor: Timothy Bourke: "A Programmer's introduction to Computer Architectures and Operating Systems" (L2), École Polytechnique, France
- Marc Pouzet and Timothy Bourke organized and reviewed L3 and M1 internships at ENS DI.

10.2.1 Supervision

- Marc Pouzet and Timothy Bourke supervised the PhD thesis of Gregoire Bussone.
- Marc Pouzet supervised the PhD thesis of Charles de Haro, together with Xavier Rival of the ANTIQUE team.
- Marc Pouzet and Timothy Bourke supervised the PhD thesis of Balthazar Patiachvili.
- Timothy Bourke supervised the L3 internship of Alexandre Douard.
- Timothy Bourke supervised the M2 internship of Balthazar Patiachvili.
- Marc Pouzet supervised the M2 internship of Henri Saudubray.

10.2.2 Juries

- Timothy Bourke was the "opponent" at the PhD defence of Nikolaus Huber at the University of Uppsala, Sweden.
- Timothy Bourke was an examiner for the PhD defence of Jordan Ischard at the University of Orléans.
- Timothy Bourke was an examiner for the PhD defence of Aurélie Kong Win Chang at the University Grenoble Alpes.

10.3 Popularization

10.3.1 Productions (articles, videos, podcasts, serious games, ...)

- Timothy Bourke participated in a [video](#) made by Esprit Sorcier TV about the Vélus project.
- Timothy Bourke was interviewed by the Pôle PSL Innovation.

10.3.2 Participation in Live events

- Timothy Bourke participated in a round-table discussion at the Institut Pasteur on "Academic Concours".

10.3.3 Others science outreach relevant activities

- Timothy Bourke made two high-school visits in the programme "1 scientifique, 1 classe: Chiche!".

11 Scientific production

11.1 Major publications

- [1] G. Baudart, L. Mandel, E. Atkinson, B. Sherman, M. Pouzet and M. Carbin. ‘Reactive probabilistic programming’. In: *PLDI 2020 - 41th ACM SIGPLAN International Conference in Programming Language Design and Implementation*. London / Virtual, United Kingdom, June 2020. doi: [10.1145/3385412.3386009](https://doi.org/10.1145/3385412.3386009). URL: <https://hal.inria.fr/hal-03051954>.
- [2] T. Bourke, V. Bregeon and M. Pouzet. ‘Scheduling and Compiling Rate-Synchronous Programs with End-To-End Latency Constraints’. In: *Leibniz International Proceedings in Informatics. 35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*. Vol. 262. 35th Euromicro Conference on Real-Time Systems (ECRTS 2023). Vienna, Austria, 3rd July 2023, 1:1–1:22. doi: [10.4230/LIPIcs.ECRTS.2023.1](https://doi.org/10.4230/LIPIcs.ECRTS.2023.1). URL: <https://inria.hal.science/hal-04149828>.
- [3] T. Bourke, L. Brun, P.-E. Dagand, X. Leroy, M. Pouzet and L. Rieg. ‘A Formally Verified Compiler for Lustre’. In: *PLDI 2017 - 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, Barcelone, Spain, June 2017. URL: <https://hal.inria.fr/hal-01512286>.
- [4] T. Bourke, L. Brun and M. Pouzet. ‘Mechanized semantics and verified compilation for a dataflow synchronous language with reset’. In: *Proceedings of the ACM on Programming Languages* 4.POPL (22nd Jan. 2020), pp. 1–29. doi: [10.1145/3371112](https://doi.org/10.1145/3371112). URL: <https://inria.hal.science/hal-02426573>.
- [5] T. Bourke, F. Carcenac, J.-L. Colaço, B. Pagano, C. Pasteur and M. Pouzet. ‘A Synchronous Look at the Simulink Standard Library’. In: *EMSOFT 2017 - 17th International Conference on Embedded Software*. Seoul, South Korea: ACM Press, Oct. 2017, p. 23. URL: <https://hal.inria.fr/hal-01575631>.
- [6] T. Bourke, J.-L. Colaço, B. Pagano, C. Pasteur and M. Pouzet. ‘A Synchronous-based Code Generator For Explicit Hybrid Systems Languages’. In: *International Conference on Compiler Construction (CC)*. LNCS. London, United Kingdom, July 2015. URL: <https://hal.inria.fr/hal-01242732>.
- [7] T. Bourke, P. Jeanmaire and M. Pouzet. ‘Functional Stream Semantics for a Synchronous Block-Diagram Compiler’. In: *LICS 2025 - 40th Annual ACM/IEEE Symposium on Logic in Computer Science*. Singapore, Singapore, 23rd June 2025. URL: <https://inria.hal.science/hal-05107499>.
- [8] T. Bourke, B. Pesin and M. Pouzet. ‘Verified Compilation of Synchronous Dataflow with State Machines’. In: *ACM Transactions on Embedded Computing Systems*. EMSOFT 2023: 23rd International Conference on Embedded Software. Vol. 22. 5s. Hamburg, Germany, 30th Sept. 2023, 137:1–137:26. doi: [10.1145/3608102](https://doi.org/10.1145/3608102). URL: <https://inria.hal.science/hal-04201401>.
- [9] J.-L. Colaço, M. Mendler, B. Pauget and M. Pouzet. ‘A Constructive State-based Semantics and Interpreter for a Synchronous Data-flow Language with State Machines: Application to the Language Scade’. In: *ACM Transactions on Embedded Computing Systems (TECS)* 22.5s (9th Sept. 2023), Article 152: 1–26. doi: [10.1145/3609131](https://doi.org/10.1145/3609131). URL: <https://hal.science/hal-04491219>.
- [10] L. Gérard, A. Guatto, C. Pasteur and M. Pouzet. ‘A modular memory optimization for synchronous data-flow languages: application to arrays in a lustre compiler’. In: *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*. Beijing, China: ACM, June 2012, pp. 51–60. doi: [10.1145/2248418.2248426](https://doi.org/10.1145/2248418.2248426). URL: <https://hal.inria.fr/hal-00728527>.
- [11] J. C. Juega, S. Verdoolaege, A. Cohen, J. I. Gómez, C. Tenllado and F. Catthoor. ‘Patterns for parallel programming on GPUs’. In: *Patterns for parallel programming on GPUs*. Ed. by F. Magoulès. Vol. Evaluation of State-of-the-Art Parallelizing Compilers Generating CUDA Code for Heterogeneous CPU/GPU Computing. ISBN 978-1-874672-57-9. Saxe-Cobourg, 2013. URL: <https://hal.archives-ouvertes.fr/hal-01257261>.
- [12] L. Mandel, F. Plateau and M. Pouzet. ‘Static Scheduling of Latency Insensitive Designs with Lucy-n’. In: *FMCAD 2011 - Formal Methods in Computer Aided Design*. Austin, TX, United States, Oct. 2011. URL: <https://hal.inria.fr/hal-00654843>.

11.2 Publications of the year

International journals

- [13] L. Sylvestre, J. Sérot and E. Chailloux. ‘Programming Parallelism on FPGAs with Eclat’. In: *International Journal of Parallel Programming* 53.4 (30th June 2025), p. 26. DOI: [10.1007/s10766-025-00801-7](https://doi.org/10.1007/s10766-025-00801-7). URL: <https://hal.science/hal-05169350>.

International peer-reviewed conferences

- [14] T. Bourke, P. Jeanmaire and M. Pouzet. ‘Functional Stream Semantics for a Synchronous Block-Diagram Compiler’. In: *LICS 2025 - 40th Annual ACM/IEEE Symposium on Logic in Computer Science*. Singapore, Singapore, 23rd June 2025. URL: <https://inria.hal.science/hal-05107499> (cit. on p. 11).
- [15] T. Bourke, P. Jeanmaire and M. Pouzet. ‘Une sémantique de Kahn mécanisée pour les machines à états’. In: *JFLA 2026 – 37es Journées Francophones des Langages Applicatifs*. Vol. JFLA 2026 – 37es Journées Francophones des Langages Applicatifs. Oberbronn, France, 27th Jan. 2026. URL: <https://hal.science/hal-05428043>.
- [16] J.-J. Chen, M. Günzel, D. Dasari, M. Becker, E. A. Lee and T. Bourke. ‘Special Sessions - Predictable Timing Behavior in Distributed Cyber-Physical Systems’. In: *EMSOFT ’25: Proceedings of the International Conference on Embedded Software*. EMSOFT 2025 - ACM International Conference on Embedded Software. Taipei, Taiwan: ACM, 5th Dec. 2025, pp. 23–32. DOI: [10.1145/3742874.3757086](https://doi.org/10.1145/3742874.3757086). URL: <https://inria.hal.science/hal-05444954> (cit. on p. 12).

11.3 Cited publications

- [17] A. Benveniste, T. Bourke, B. Caillaud, J.-L. Colaço, C. Pasteur and M. Pouzet. ‘Building a Hybrid Systems Modeler on Synchronous Languages Principles’. In: *Proceedings of the IEEE. Design Automation for Cyber-Physical Systems* 106.9 (Sept. 2018), pp. 1568–1592. DOI: [10.1109/JPROC.2018.2858016](https://doi.org/10.1109/JPROC.2018.2858016). URL: <https://hal.inria.fr/hal-01879026> (cit. on p. 6).
- [18] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic and R. de Simone. ‘The synchronous languages 12 years later’. In: *Proceedings of the IEEE* 91.1 (Jan. 2003) (cit. on p. 6).
- [19] G. Berry. ‘Real Time programming: Special purpose or general purpose languages’. In: *IFIP Congress*. Elsevier Science Publishers, 1989, pp. 11–17 (cit. on p. 6).
- [20] S. Blazy, Z. Dargaye and X. Leroy. ‘Formal Verification of a C Compiler Front-End’. In: *FM 2006: Int. Symp. on Formal Methods*. Vol. 4085. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 460–475. URL: <http://gallium.inria.fr/~xleroy/publi/cfront.pdf> (cit. on p. 10).
- [21] T. Bourke, J. Inoue and M. Pouzet. ‘Sundials/ML: connecting OCaml to the Sundials numeric solvers’. In: *EPTCS* 285 (2018), pp. 101–130. DOI: [10.4204/EPTCS.285.4](https://doi.org/10.4204/EPTCS.285.4). URL: <http://www.tbrk.org/papers/abstracts.html#eptcs2018> (cit. on p. 13).
- [22] T. Bourke, J. Inoue and M. Pouzet. ‘Sundials/ML: interfacing with numerical solvers’. In: *ACM Workshop on ML*. ACM. Nara, Japan, Sept. 2016. URL: <http://www.tbrk.org/papers/abstracts.html#ml16> (cit. on p. 13).
- [23] T. Bourke, B. Pesin and M. Pouzet. ‘Verified Compilation of Synchronous Dataflow with State Machines’. In: *EMSOFT 2023: 23rd International Conference on Embedded Software*. Vol. 22. 5s. Hamburg, Germany, Sept. 2023, 137:1–137:26. DOI: [10.1145/3608102](https://doi.org/10.1145/3608102). URL: <https://inria.hal.science/hal-04201401> (cit. on p. 11).
- [24] P. Caspi and M. Pouzet. ‘Synchronous Kahn Networks’. In: *ACM SIGPLAN International Conference on Functional Programming (ICFP)*. Philadelphia, Pennsylvania, May 1996 (cit. on p. 6).
- [25] J.-L. Colaço, G. Hamon and M. Pouzet. ‘Mixing Signals and Modes in Synchronous Data-flow Systems’. In: *ACM International Conference on Embedded Software (EMSOFT’06)*. Seoul, South Korea, Oct. 2006 (cit. on p. 11).

- [26] J.-L. Colaço, B. Pagano and M. Pouzet. ‘A Conservative Extension of Synchronous Data-flow with State Machines’. In: *ACM International Conference on Embedded Software (EMSOFT’05)*. Jersey city, New Jersey, USA, Sept. 2005 (cit. on p. 11).
- [27] J.-L. Colaço, B. Pagano and M. Pouzet. ‘Scade 6: A Formal Language for Embedded Critical Software Development’. In: *TASE 2017 - 11th International Symposium on Theoretical Aspects of Software Engineering*. Nice, France, Sept. 2017, pp. 1–10. URL: <https://hal.inria.fr/hal-01666470> (cit. on p. 6).
- [28] *The Coq proof Assistant*. <http://coq.inria.fr>. 2019 (cit. on p. 10).
- [29] N. Eén. ‘Translating Pseudo-Boolean Constraints into SAT’. In: *J. Satisfiability, Boolean Modelling and Computation* 2.1–4 (Feb. 2006), pp. 1–26. doi: [10.3233/SAT190014](https://doi.org/10.3233/SAT190014) (cit. on p. 12).
- [30] G. Berry. ‘Real time programming: Special purpose or general purpose languages’. In: *Information Processing* 89 (1989), pp. 11–17 (cit. on p. 6).
- [31] A. Hindmarsh, P. Brown, K. Grant, S. Lee, R. Serban, D. Shumaker and C. Woodward. ‘SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers’. In: *ACM Transactions on Mathematical Software* 31.3 (Sept. 2005), pp. 363–396 (cit. on p. 13).
- [32] P. Jeanmaire. ‘A denotational semantics for a verified synchronous compiler’. Theses. Université PSL (Paris Sciences & Lettres), Dec. 2024. URL: <https://hal.science/tel-04885682> (cit. on p. 11).
- [33] G. Kahn. ‘The semantics of a simple language for parallel programming’. In: *IFIP 74 Congress*. North Holland, Amsterdam, 1974 (cit. on p. 6).
- [34] G. Kahn and D. B. MacQueen. ‘Coroutines and Networks of Parallel Processes’. In: *IFIP Congress*. 1977, pp. 993–998 (cit. on p. 6).
- [35] X. Leroy. *The CompCert verified compiler*. 2009. URL: <http://compcert.inria.fr/doc/index.html> (cit. on pp. 10, 11).
- [36] L. Mandel and M. Pouzet. ‘ReactiveML, a Reactive Extension to ML’. In: *ACM International Conference on Principles and Practice of Declarative Programming (PPDP)*. Lisboa, July 2005 (cit. on p. 6).
- [37] B. Pautet. ‘Memory Specification in a Data-flow Synchronous Language with Statically Sized Arrays’. PhD thesis. Paris, France: PSL Université, Dec. 2023 (cit. on p. 13).
- [38] C. Paulin-Mohring. ‘A constructive denotational semantics for Kahn networks in Coq’. In: *From Semantics to Computer Science: Essays in Honour of Gilles Kahn*. Ed. by Y. Bertot, G. Huet, J.-J. Lévy and G. Plotkin. Cambridge, UK: Cambridge University Press, 2009, pp. 383–413. URL: <https://hal.inria.fr/inria-00431806/document> (cit. on p. 11).
- [39] B. Pesin. ‘Verified Compilation of a Synchronous Dataflow Language with State Machines’. PhD thesis. PSL Research University, Oct. 2023. URL: <https://velus.inria.fr/phd-pesin/thesis.pdf> (cit. on p. 11).
- [40] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002 (cit. on p. 6).
- [41] M. Sheeran. ‘muFP, a language for VLSI design’. In: *ACM Conference on LISP and Functional Programming*. Austin, Texas, 1984, pp. 104–112 (cit. on p. 6).
- [42] Z. Wan and P. Hudak. ‘Functional Reactive Programming from First Principles’. In: *International Conference on Programming Language, Design and Implementation (PLDI)*. 2000 (cit. on p. 6).